Exploratory Analytics for RDF Graphs

Shu Shang

Master Research Internship Report

Exploratory Analytics for RDF Graphs

Author:
Shu SHANG

Internship Supervisor:
Yanlei Diao
Ioana Manolescu

Scientific Supervisor:
Albert Bifet
Yannis Manoussakis

A thesis submitted in partial fulfillment of the requirements for the degree of
Master of Science

Université Paris-Saclay
Computer Science - "Data & Knowledge"

September 7, 2017

Acknowledgements

Firstly, I would like to thank my internship advisors Prof. Yanlei Diao and Prof. Ioana Manolescu, for giving me this opportunity to do my internship in the Inria Cedar team, and their kind guidance and help during my internship.

I would like to thank my scientific advisors Albert Bifet and Yannis Manoussakis for his advice and help during my internship.

Secondly, I would like to thank all the researchers and students in Inria Cedar team. They have been providing both happiness and kind help during my internship.

Contents

1 Introduction

1.1 Outline

The document reports on my internship integrated into my second year Master studies in the "Data&Knowledge" program of Université Paris Saclay. The internship was carried out within the CEDAR team at Inria-Saclay, starting from March of 2017 and lasting 24 weeks. The internship topic is exploratory analytics for RDF data.

The report is organized as follows:

- Section 1: Introduction
- Section 2: Research environment
- Section 3: Preliminaries
- Section 4: Previous work
- Section 5: Problem statement
- Section 6: Algorithm
- Section 7: Implementation and results
- Section 8: Conclusion and perspectives for future work

1.2 Context and motivation

The "Web of data" vision behind the initial World Wide Web project has found its most recent incarnation through the Semantic Web. More and more data are generated as *triples* using RDF (the Resource Description Framework), which is the W3C standard for describing interconnected resources. To explore RDF data, the SPARQL query language has been defined [14] and recently enriched to support complex querying using regular path expressions, grouping and aggregation etc.

RDF graph may be very large and their structure is complex and heterogeneous, making it very difficult to find valuable insights from it. In this work, we seek to devise a framework to facilitate RDF analytics. Firstly, we are interested in aggregate queries over RDF graph, in the spirit of relational data warehouse. For each graph, we are looking for properties which can be used as *dimensions*, for each dimension, a set of *measures* are proposed, along with applicable *aggregation functions*. Then, a mechanism to evaluate the aggregate queries should be defined. In this work, we evaluate the queries by order of their interestingness. Moreover, the evaluation results should be visualized. In this work, we plot the evaluation results of aggregate queries as bar charts, and show them to the user.

1.3 Dagger: Digging for Interesting Aggregates in RDF Graphs

Based on the motivation and context above, we developed **Dagger**, a system which automatically identifies the most interesting insights in a given RDF graph G, defined as RDF *aggregate queries* evaluated over G. Dagger ranks such insights in the decreasing order of their interestingness, evaluates and plots the most interesting ones as bar charts, and shows them to the user. In Chapter 5 and Chapter 6, we formalize the problem and present Dagger's solution.

A paper based on this work has been accepted as a demo paper at the International Semantic Web Conference (ISWC) 2017.

2 Research environment

We describe below the research environment of the internship: the CEDAR team of Inria.

2.1 Inria

Inria, the French Institute for Research in Computer Science and Automation (Institut national de recherche en informatique et en automatique, in French), is a French national research institution focusing on computer science and applied mathematics. It aims at promoting "scientific excellence for technology transfer and society". Research work is organized based on "project-teams", by gathering researchers from similar domains.

Inria has totally 8 research centers (in Bordeaux, Grenoble-Inovallée, Lille, Nancy, Paris-Rocquencourt, Rennes, Saclay, and Sophia Antipolis) and also collaborates with other ace-ademic research outside these centers, such as the LRI lab of Université Paris Sud or the LIX lab of Ecole Polytechnique. Research teams are divided into 5 different fields:

- Applied Mathematics, Computation and Simulation
- Algorithmics, Programming, Software and Architecture
- Networks, Systems and Services, Distributed Computing
- Perception, Cognition and Interaction
- Digital Health, Biology and Earth

CEDAR belongs to "Perception, Cognition and Interaction", under the sub-category of "Data and Knowledge Representation and Processing".

Inria employs 3800 people, among which there are 1300 researchers, 1000 Ph.D. students and 500 postdoctorates.

2.1.1 Inria: organization

The Chairman and CEO of Inria is Anotoine Petit. Prior to that, he is a PhD and processor in Computer Science, and a formerly Deputy Managing Director of Inria.

The management team has totally 18 people, including the president of different research centers. Chart 2.1 illustrated the general organization of Inria.

The organization of Inria also includes an administrative committee, two scientific committees, and the regular visit of an outside committee.

2.1.2 Inria: key figures

Below, we collect some recent key figures of Inria (Updated in July 2017), which illustrate different aspects of the institute in 2016.

- Scientific activities
  - 183 Inria project-teams
  - 4,500 scientific publications per year
- Industrial relations
  - 410 active patents
  - 126 software programmes registered with France's Software Protection Agency in 2016
  - 44 start-ups Inria since 2010
- Structure
  - 8 research centres located throughout France (Paris, Rennes, Sophia Antipolis, Grenoble, Nancy, Bordeaux, Lille and Saclay) and a head office in Rocquencourt, near Paris.
- Human resources
  - 2,400 members from 102 different countries
  - 1,200 PhD students
- Budgetary resources
  - Total budget: €231M
  - Proportion self-financed: 25%

2.2 Inria Saclay

Inria Saclay is the center that I am affiliated. It was created at 1st January 2008. The center has 450 scientists and 100 research support stuff. It has 31 research teams, 23 of which are joint ventures with other establishments like CEA Saclay and École Polytechnique. Many of co-joint establishments are located at "Plateau de Saclay".

In addition to its natural collaborations within the European Union, Inria Saclay also participates in a large number of collaboration projects aorund the world. These mixed projects are created through the personal contacts of individual researchers and teams, or through bi- or multi-lateral agreements. Research partners are from establishments or universities like University of California, Berkeley and Stanford University.

2.3 The Cedar team

I was integrated into the Cedar team of Inria-Saclay. It was created in 2016 and it is joint team between Inria Saclay and LIX (CNRS – UMR 7161 and Ecole Polytechnique). The team focus on "Rich data analytics at cloud scale".

The team has 3 permanent members: Ioana Manolescu (INRIA senior researcher, team leader), Yanlei Diao (École Polytechnique professor, on a joint position with Télécom ParisTech) and Michaël Thomazo (INRIA junior researcher, associate team leader). The team has also 2 associated members: François Goasdoué (Professor at Univ. Rennes 1) and Xavier Tannier (LIMSI, CNRS/U. Paris Sud).

At present, the team has 7 Ph.D. students, 2 engineers, 2 interns (including me) and 1 post-doc.

2.3.1 Research themes

The research projects of the CEDAR team are focused on two themes:

- Exploiting parallel data processing infrastructures. The direction aims at developing highly scalable, parallel Big Data storage and processing tools.
- Seeking new paradigms of user interaction with Big Data. The direction aims at devising new tools for mining values from Big data, based on exploratory query, analytics for semantic graphs etc.

2.3.2 Current projects

ContentCheck

The project aims at designing new models, algorithms and tools for **fact checking**.

Fact-checking is the task of assessing the factual accuracy of claims, typically prior to publication. Modern fact-checking is faced with a triple revolution in terms of scale, complexity, and visibility, as claims and background knowledge are increasingly digital.

This project brings together academic labs with expertise in data management, natural language processing, automated reasoning and data mining, and a fact-checking team of journalists from a major french web media.

The team working on this project aims to establish fact-checking as a data management problem, endows it with sound foundations, designs and deploys new algorithms for automating fact-checking, and validates them by close interaction with the journalists.

WebClaimExplain

It is a co-joint project with AIST (Advanced Industrial Science and Technology) in Japan.

The recent evolution of the Internet, such as the emergence of social networks, open data and sensor networks, have made it challenging for people or businesses to cope with the information deluge. Virtually any decision, from voting to calling the doctor or buying stocks, is based on facts we find around us, and increasingly on the Internet.Facts are published by individuals (e.g. journalists, lobbyists, social users), organizations (e.g. public relations agencies, media outlets, governments), or machines (e.g. news generators, disaster monitors). The goal of this research is to create

tools to find explanations for facts and verify claims made online. Several challenges emerge in the pursuit of such a goal:

- Statements are generally made in natural languages, which are notoriously hard to process algorithmically;
- Even when statements are available in machine-processable form, determining the "truth" of claims is difficult because of the inherent lack of contextual information (time, space, political views, belief systems, etc.);
- Assuming "sufficient" context is available, one still needs to use external sources and inference mechanisms to draw conclusions — if the trustworthiness of such sources and rules are subject to caution, this may lead to weak or simply wrong conclusions. In this respect, it is clear that the process cannot be fully automated. The main focus of our work will be explanation finding via trusted sources, based on the observation that one can only trust a statement if he/she can explain it through rules and proofs that can themselves be trusted.

RDFSummary

This project seeks to devise a query-oriented tool for summarization of RDF graphs. The problem of RDF summarization, meaning: given an RDF graph, find an RDF summary graph which summarizes the input dataset as accurately as possible, while being possibly orders of magnitude smaller than the original graph. Such a summary can be used in a variety of contexts: to help an RDF application designer get acquainted with a new dataset, as a first-level user interface, or as a support for query optimization as traditionally the case in semi-structured graph data management etc.

Different kinds of summaries are proposed based on the treatment of typed resources and degree of similarity of property sets between resources, including **weak**, **typed-weak**, **strong** and **typed strong** summary.

A set of sample summaries of different RDF datasets can be accessed at: `https://team.inria.fr/cedar/projects/rdfsummary/`.

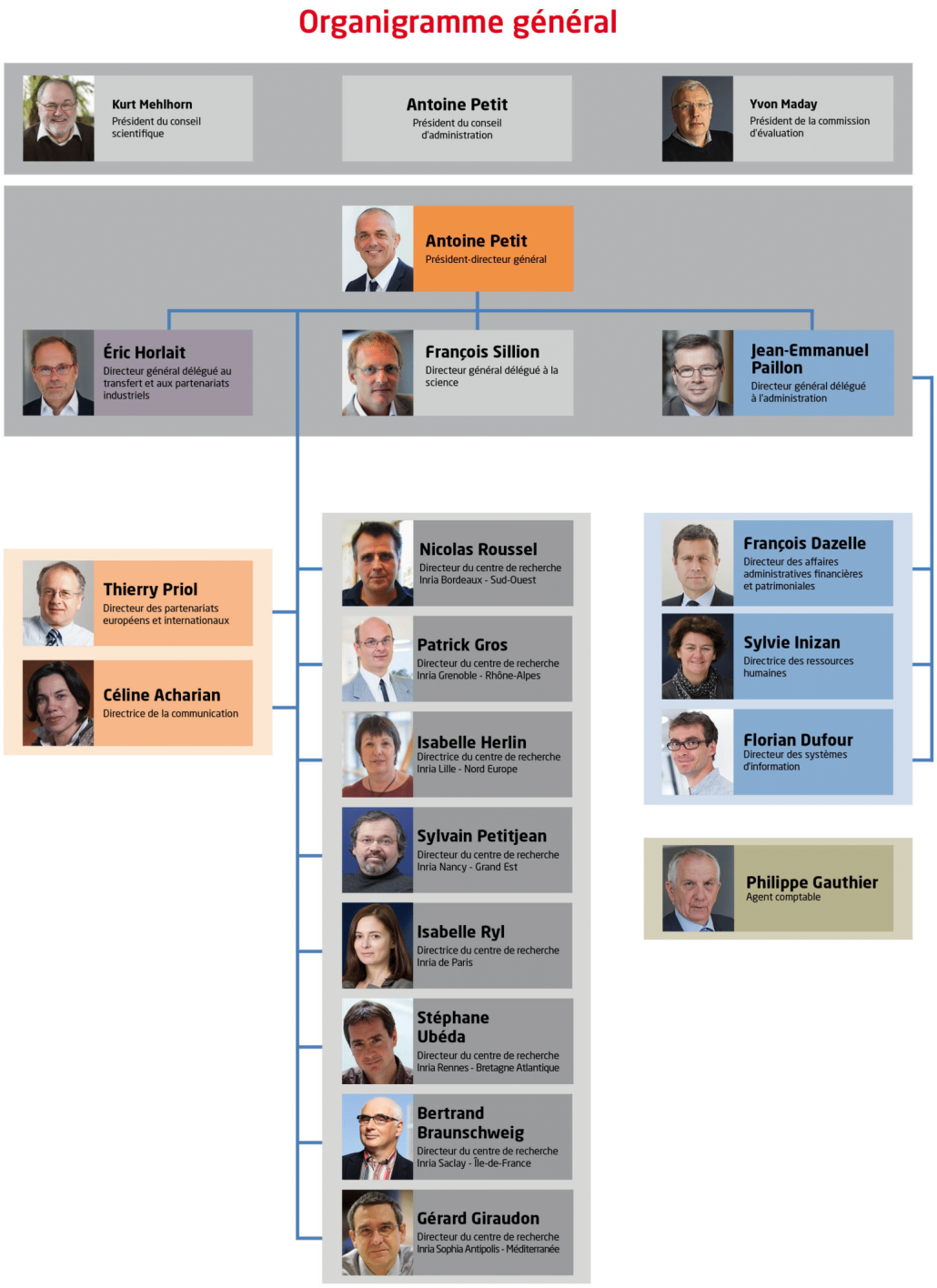


FIGURE 2.1: Organization of Inria

3 Preliminaries

3.1 Semantic Web

The **semantic web** is an extension of the Web through standards by the World Wide Web Consortium (W3C). The term was coined by Tim Berners-Lee, aiming at creating a new generation of web (Web 3.0) on which the data can be processed and understood by machines.

Currenlty, the World Wide Web is based mainly on HTML (Hypertext Markup Language) documents, a markup convention that is used for coding web page. An HTML page is composed by metadata tags, along with the data to be expressed. An HTML page can be rendered by a web browser. Nowadays, web technologies are among the most important technologies of Information, large amounts of information are expressed using web pages. However, HTML has no capability to express semantics. For instance, by reading an HTML page, we may know that the profession of a person called Bob is "musician". However, this information are placed in different tags of a plain HTML page with no explicit relationships with each other. Consequently, we cannot know that Bob is a musician and a musician is a person. More specifically, machines cannot obtain or infer semantics from a plain HTML page.

The semantic web is designed to make the web understandable. It involves publishing in languages specifically designed for data: Resource Description Framework (RDF), Web Ontology Language (OWL) and and Extensible Markup Language (XML). HTML describe documents and links between them. RDF, OWL and XML, by contrast, can describe things in the real world such as People, Position or Country.

The set of technologies of semantic web amis at replacing the current content of Web documents. By publishing using technologies of the semantic web, content is combined with meaning. A machine can thus process and infer knowledge over the content, using processes similar to human deductive reasoning and inference.

3.2 RDF, RDF graph

RDF (the Resource Description Framework) is the W3C standard for describing interconnected resources. It is designed as a metadata data model. It is in the form of **subject-predicate-obejct** expressions, known as triples. The subject denotes the resource, the predicate denotes aspects or characteristics of the resource, and expresses a relationship between the subject and the object.

For example, one way to represent the notion "Bob has the profession musician" in RDF is the triple:

- a subject denoting Bob

- a predicate denoting "has the profession"
- an object denoting "Musician"

This standard for describing resources is a key component of Semantic Web project. By using RDF through the World Wide Web, software can process and exchange machine-readable data, making it possible to build more intelligent applications like Knowledge Graphs.

RDF can be expressed using different formats, based on different ways for storing and transmitting data. The most important formats are: Notation3, Turtle and N-Triples.

An **RDF graph** is a set of triples, typically denoted as (s, p, o), where s is the subject, p is the predicate (also called property) and o is the object, or value of the property p of s. RDF graph is complex and heterogeneous; in particular, a resource can have several values for a property. For instance, resource representing journal articles may have several values for the property *creator*.

Below is an example of RDF graph:

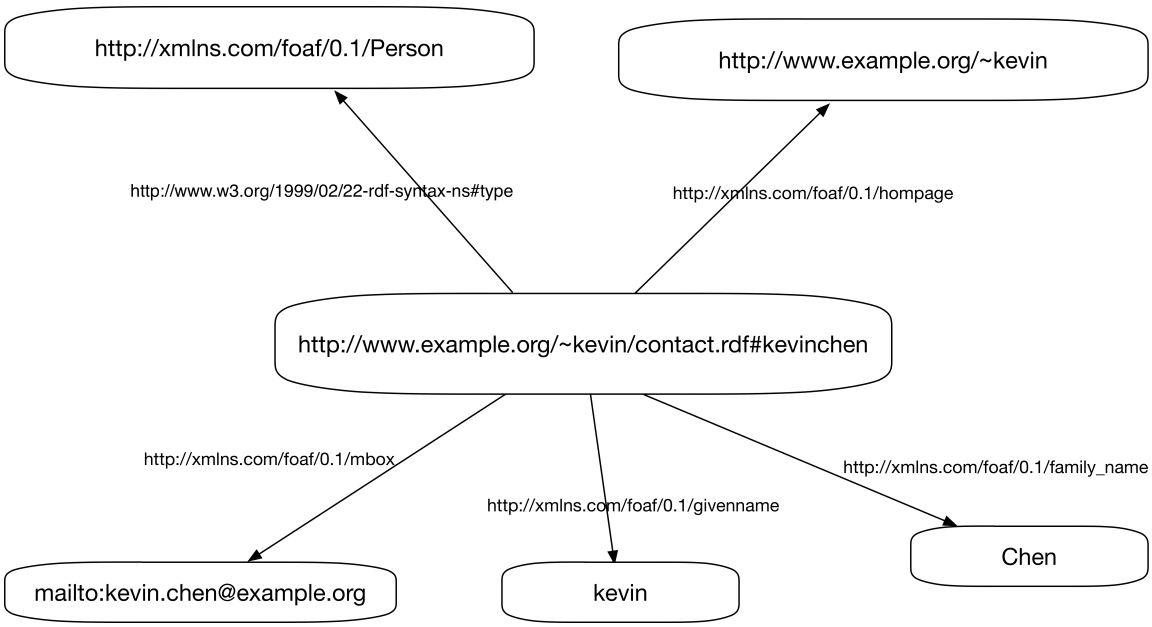


FIGURE 3.1: An example of RDF graph

As the example illustrates, subjects and objects are presented as node, whereas a property is represented as an edge between two nodes. From this example, we know that: “Kevin Chen” is a Person, his given name is “Kevin” and his family name is “Chen”, his homepage as well as his mail box address. It should noted that nodes and edges can be URI (Unique Resource Identifier)[1] such as:

`http://www.example.org/~kevin/contact.rdf#kevinchen`

is a URI (denoted as s for short) for representing a person called Kevin Chen.

[1]URIs are defined in the specifications available at: https://www.w3.org/Addressing/. Technically speaking, the most current specification applying is that of International Resource Identifiers (IRIs); for the purpose of this document, the difference between URIs and IRIs can be ignored.

In contrast, property objects can be URIs or literals (which can be viewed as constants; see below).

The special property: `http://www.w3.org/1999/02/22-rdf-syntax-ns#type` (*type*, for short) allows a form of resource typing. In this example, we know that (s,type,*Person*), stating that the resource represented by s is of type *Person*.

A resource may have one or several types, or it may lack types. Property value (the object) can be URIs, blank nodes (a special form of unknown/unspecified nodes), Strings, Integers, float numbers, dates etc. Values may be typed explicitly, as in "12.5^^xsd:decimal", but may also appear simply as "12.5". In the example above, the property value for given name and family name is Strings, whereas this information may not appear in the original data.

From an RDF graph and some semantic constraints, we may derive *implicit triples*. For instance, given:

$$p1\ type\ SmartPhone$$

and

$$SmartPhone\ subclassOf\ Phone$$

we can derive $p1$ *type Phone*. By applying such inference steps based on data triples and semantic rules, we can get an RDF graph which contains both explicit and implicit triples. We name the process of applying entailment rules to an RDF graph the *saturation* of an RDF graph. An RDF graph is saturated if it contains all its implicit triples.

In this work, we will consider that the RDF graphs we work with are already saturated, and will not focus any further on inference.

3.3 Storing RDF: triple stores

RDF data can be stored in various kinds of databases, including relational databases, graph databases, document-oriented database etc. There existing also a large amounts of methods for storing RDF data in a traditional way [9]. A syste capable of storing RDF data is commonly called a *triplestore* (also called RDF store). It is a special database for storing and retrieval RDF data (triples) composed of subject-predicate-object triples, through semantic queries like SPARQL.

Triplestore can be implemented in various ways. They can be designed and implemented from scratch, or they can also reuse existing relational database engines like PostgreSQL or NoSQL engines. By using existing relational database engines, triplestores can be constructed with little effort while reusing existing powerful functions of relational databases like OLAP operations. When building triplestores over existing relational database engines, one has to handle the translation between the RDF-specific SPARQL query language and SQL, the language supported at the relational database level.

3.4 RDF query language: SPARQL

SPARQL[14] is the query language for RDF. SPARQL was put forward by the World Wide Web Consortium (W3C), which is the main standards organization for the World Wide Web. It is a key technology used in most Semantic Web applications; the current version of SPARQL is 1.1. From version 1.0, SPARQL has become an official recommendation of the W3C.

Using SPARQL, we can express queries across diverse data sources, including traditional triplestore and RDF views in a middleware style. SPARQL has remarkable expressive power. SPARQL queries can follow a specific path over an RDF graph by conjunctions and disjunctions, making it very useful to query complex graph structure available in an RDF graph.

For example, given a simple RDF graph:

```
subject: <http://example.org/article/article1>
predicate: <http://purl.org/dc/elements/1.1/title>
object: "An introduction to SPARQL"
```

A simple SPARQL query to find the title of an article is:

```
SELECT ?title
WHERE
{
  <http://example.org/article/article1> <http://purl.org/dc/elements/1.1/title> ?title
}
```

The query result will be:

```
title  "An introduction to SPARQL"
```

Most forms of SPARQL queries contain a set of triple patterns called a *basic graph pattern (BGP)*. A well-known subset of SPARQL consists of (unions of) BGP queries, also known as SPARQL conjunctive queries. A BGP is a set of *triple patterns*, or triples in short. Triple patterns are like RDF triples except that each of the subject, predicate and object may be a *variable*. A basic graph pattern is said to *match* a subgraph of the RDF data.

For example, the query above consists of two parts: the SELECT clause identifies the variables to appear in the query results, and the WHERE clause provides the basic graph pattern to match against the data graph. The basic graph pattern in this example consists of a single triple pattern with a single variable (?title) in the object position.

It is useful to view each triple pattern atom in the body of a BGP query as a generalized RDF triple, where variables may appear in any position of subject, predicate, subject. This idea leads to a graph notation for BGP queries, which can be seen as a generalizations of RDF graphs. For instance, given a query in the form of relational calculus:

```
q(x, y, z):- x hasGivenName y, x z Person
```

can be represented by the graph:

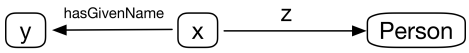


FIGURE 3.2: An example of graph notation of BGP query

The evaluation of a query can be then viewed as replacing every occurrence of a variable in a triple pattern by its real value. It should be noted that, as the original RDF graph may contain implicit triples, the evaluation of a query should be done over the saturation of a given RDF graph.

From version 1.0 onward, SPARQL can support some complex queries, including union, optional query parts and filters. SPARQL 1.1 has been released in March 2013, with many new features, including subqeries, value assignment, path expression and aggregate functions.

One of the most important features is aggregate functions. Below is an example of SPARQL aggregate query:

Let `http://example.org/kevin` be an URI and assume a graph is published at that address, containing information about Kevin and his social contact information. Below is an illustration of a sub-portion of the original graph:

```
@prefix foaf: <http://xmlns.com/foaf/0.1/> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .

<http://example.org/kevin#me> a foaf:Person .
<http://example.org/kevin#me> foaf:name "Kevin" .
<http://example.org/kevin#me> foaf:mbox <mailto:kevin@example.org> .
<http://example.org/kevin#me> foaf:knows <http://example.org/alice#me> .
<http://example.org/alice#me> foaf:knows <http://example.org/kevin#me> .
<http://example.org/alice#me> foaf:name "Alice" .
<http://example.org/alice#me> foaf:knows <http://example.org/pascale#me> .
<http://example.org/pascale#me> foaf:knows <http://example.org/alice#me> .
<http://example.org/pascale#me> foaf:name "Pascale" .
<http://example.org/alice#me> foaf:knows <http://example.org/snoopy> .
<http://example.org/snoopy> foaf:name "Snoopy"@en .
```

In the above, **foaf** is an ontology definition, often called "friend-of-a-friend" ontology. Assuming this graph has been loaded into a SPARQL service (an HTTP endpoint that can be accessed by SPARQL queries), SPARQL 1.1 can be used to formulate queries ranging from simple graph pattern matching to complex queries. For instance, one can query the name of persons and the number of their friends using the SPQRQL query below:

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
SELECT ?name (COUNT(?friend) AS ?count)
WHERE {
    ?person foaf:name ?name .
    ?person foaf:knows ?friend .
} GROUP BY ?person ?name
```

The result of evaluating a SPARQL query can be obtained in different formats, including XML, JSON, CSV and TSV.

4 Previous work

Tools comparable to Dagger have been developed in relational data warehouses, when the possible aggregation dimensions and measures are known [15, 16]; [15] also investigates how to efficiently derive an interesting aggregate from another. In contrast, Dagger applies on a heterogeneous RDF graph in which no facts, dimensions, and measures are known, and recommends candidate queries through a set of effective heuristics. [4] selects interesting cuboids from an "aggregated graph", pre-built by analysts from relational or any other kind of data, counting associations between two resources of the same kind (e.g., social network interactions between users), while Dagger applies directly on (any) RDF graph. Dagger complements many other RDF exploration techniques such as mining, summarization, faceted search, statistics extraction etc. [11, 12].

Below, we present several works from which Dagger has gained much inspirations.

4.1 RDF analytics: lenses over semantic graphs

The development of the semantic web (and its core component RDF) brings new requirements for RDF analytical tools and methods. Recent work [6] has proposed a framework for analyzing RDF graphs, in the spirit of relational data warehouse, but redesigned in order to adapt to the specifications of RDF data model. Some ideas of my internship are based on this work.

This paper, from the beginning, defined the core concepts and tools in the context of RDF data, including analytical schema (AnS), analytical query (AnQ) and OLAP-style operations on RDF graph.

Recall the introduction of RDF graph in Chapter 3. Given an RDF graph in Figure 4.1:

we can issue semantic SPARQL queries (especially BGP queries introduced in 3), asking questions like:

How much is the $product_1$?

More complex queries, for instance:

How many sites each blogger has posted, classified by the blogger's age and cite?

are not supported in the early framework of SPARQL (version 1.0), especially in a relational data warehouse setting.

The paper thus defined *analytical schemas*, through which we analyze an RDF graph. An analytical schema (AnS) is a labeled directed graph. From a classical data warehouse analytics perspective, each node in an analytical schema corresponds a set of

$$
\mathtt{G} = \begin{array}{l}\{\text{user}_1 \text{ hasName “Bill”, user}_1 \text{ hasAge “28”, user}_1 \text{ friend user}_3,\\ \text{user}_1 \text{ bought product}_1, \text{product}_1 \text{ rdf:type SmartPhone},\\ \text{user}_1 \text{ worksWith user}_2, \text{user}_2 \text{ hasAge “40”}, \ldots\}\end{array}
$$

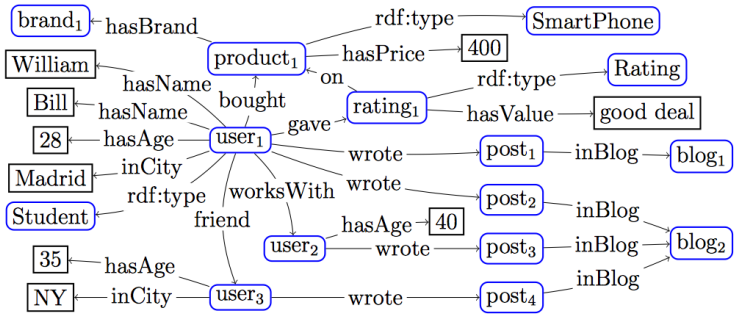


FIGURE 4.1: Running example: RDF graph (taken from the paper)

facts to be analyzed. From a Semantic Web perspective, an analytical schema node corresponds to an RDF class, while an edge connecting two nodes corresponds to an RDF property.

Below is an example of analytical schema based on the previous RDF graph example:

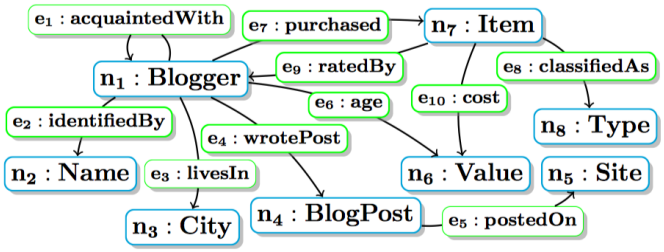


FIGURE 4.2: Sample Analytical Schema (taken from the paper)

Figure 4.2 depicts an *AnS* for analyzing bloggers and items. Each node and edge is defined by a BGP query. For instance, the node n_1 *Blogger* can be defined as:

$q(x)$:-x rdf:type Person, x wrote y, y inBlog z

In Figure 4.2, a *blogger* may have written *posts* ($e3$) which may appear on some site ($e5$). A person may also purchase items ($n7$) which can be rated ($e9$). The nodes and edges of an analytical schema define the perspective (or lens) through which to analyze an RDF graph. Based on analytical schema, the paper defined analytical schema instance, which corresponds to a data instance obtained from the original RDF graph which corresponds to the analytical schema and consists of the union of the answers to the queries defining the analytical schema nodes and edges.

The core of this framework is analytical queries, specifying: a set of resources to be analyzed, the set of dimensions along which to analyze the resources, and optionally aggregation functions to be applied for each set of resources having the same value along the dimensions. Below is an example of analytical query:

$$< c(x, y_1, y_2),\ m(x, z),\ count >$$
where the classifier and measure (queries) are defined by:
$$c(x, y_1, y_2) \text{ :- } x \text{ age } y_1,\ x \text{ livesIn } y_2$$
$$m(x, z) \text{:- } x \text{ wrotePost } y,\ y \text{ postedOn } z$$

The query above asks for the number of sites where each blogger posts, classified by the blogger's age and city.

It should be noted that in the framework proposed by the paper, an *analytical schema* (AnS) models a set of classes and properties (corresponding to nodes and edges), which are the only ones visible for further RDF analytics. In other words, analytical queries are formulated against the AnS, rather than the original base data. Consequently, the paper actually defined a complete approach for analyzing RDF graphs.

Moreover, this framework also supports On-Line Analytical Processing (OLAP), which is highly useful in the context of data warehouses. Traditional OLAP concepts and operations can thus be applied to RDF setting. In particular, traditional OLAP operations allow transforming a cube into another, like slice, dice etc. In the abovemetioned work, paper, these traditional OLAP operations on cubes can be viewed as AnQ rewritings. The details can be found in this paper and here we will not go into detail. A further in-depth reading [2] give a more complete introduction to OLAP operations on RDF graph based on analytical schemas and analytical queries.

Our work is based on this previous work. More specifically, the aggregation queries generated by Dagger are a subset of RDF analytical queries introduced in [6] and also of W3C's SPARQL 1.1 aggregation queries. The detail will be introduced in Chapters 5 and 6.

4.2 SeeDB: Efficient data-driven visualization recommendations to support visual analytics

Another work [16] based on data data visualization was also a source of inspiration. The problem they consider is:

> How to help users (domain experts) discover a dataset more efficiently and interactively?

This paper proposed an visualization recommendation engine called **SeeDB**, which is designed to facilitate data exploration, especially data with high dimension properties (high-dimensional dataset).

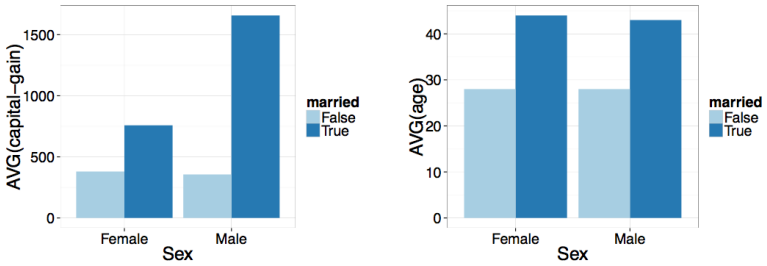
To design such a recommendation engine, two problems arise: **scale** and **utility**. Scale means that the engine should be able to compute the recommendation within interactive time scale (a high latency will give users a bad experience). Utility means that the engine should be able to rank visualizations according to their likely interests for the expert.

For scalability, the authors proposed two optimization mechanisms: 1. **sharing-based optimization** 2. **pruning based optimization**. For utility, the authors proposed a utility metric based on deviation from some *reference data*.

The authors introduced an example where they study demographics about millennials. The researchers want to find some interesting insight in the relation between two

different indicators, such as the **marital status** and **average income**, or **marital status** and **average age** etc.

Based on this data, the authors envision getting two visualizations shown in Figure 4.3.



(a) Interesting Visualization *(b) Uninteresting Visualization*

FIGURE 4.3: Relation between marital status and income/age (taken from the paper)

The left figure is more interesting. As we can see: although the difference between the average income for un-married males and females is not important, the difference of average income between married males and females is huge. However, the right-hand side figure is less interesting, as it shows less deviation for males and females with different marital status.

This comes to the definition of the **reference dataset** we mentioned above. In the first example (left-hand side), the target dataset is married adults (including male and female) and the reference dataset is unmarried adults. The deviation is thus defined between the target and the reference dataset. This is an example to illustrate **Utility** metric proposed in the paper.

The system architecture and experiments of SEEDB will not be detailed here. This work basically give us an idea about the metric **interesting**. What makes a plot interesting? How to define the interesting metric? Based on this, Dagger actually proposed a set of interesting metrics, including variance, skewness and kurtosis, which will be introduced in Chapter 5.

4.3 AIDE: An Active Learning-based Approach for Interactive Data Exploration

The topic of my internship is exploratory RDF analytics. We seek to devise a framework supporting exploratory analytics, based on the requirements that:

1. User feedback should be elicited (also called human-in-the-loop);
2. The tool should return analyzed results within interactive time scales.

Recent work [8] proposed a novel automatic data exploration framework called **AIDE**. The idea comes from a bottleneck when using traditional data management systems. Traditional data management systems assume that when user formulate a query, they (*i*) have a good knowledge about the schema, meaning and content of the database

and (*ii*) are certain that this particular query is the one they wanted to ask. More generally, traditional data management systems are designed for domain experts.

However, we are now living in the age of Big Data. Data are being collected and stored at an unprecedented rate. Applications driven by data are largely needed, including *Interactive Data Exploration* (**IDE**, in short). IDE helps users explore underlying dataset by asking sample queries. Based on user's feedback, the space for exploration can be reduced, thus formulating more accurate queries.

This paper proposed an *Automatic Interactive Data Exploration (AIDE)* framework, helping user discover data relevant to their interests. Query formulation, query processing and results reviewing are merged into one step. In this tool, users are exposed to a "conversation-like" interface. The system proposes data sample to the user, and based on the user's feedback, it builds a user model which predicts data objects matching their interests in the next iterations. The user model will be enriched in the next iterations in a similar way.

AIDE integrates the *active learning* paradigm and *sample acquisition* through a set of exploration iterations. Active learning is a sub-domain of machine learning, in which the learning algorithm is capable to interactively query the user (or some other information source). This situation comes to the stage especially when the underlying data is unlabeled. For instance, in the previous example, the label (relevant/unrelevant) is not associated with each data sample in the beginning. Active learning algorithm will then query the user and label the data in an incremental way.

Our objective is to devise such a analytical tool in the context of RDF data. In the beginning of the internship, we decided to devise an automatic mining algorithm in the first place, meaning that we will not integrate users in the application. However, the work can be continued in the future in an exploratory setting.

4.4 Data profiling

Data profiling is the process of examining the data available in an existing data source (e.g. database or a file) and collecting statistics and information about that data [13]. The purpose includes discovering of metadata of source database such as value pattern and distributions, key constraints (candidates) and functional dependencies etc.

Although existing data source structures used in enterprise may be well defined, there are scenarios which can benefit from data profiling, as follows. Consider for instance that a database administrator (DBA) created a database for a specific purpose in the past, and the data has already been dumped after being used during some period. After several years, the company may suddenly be interested to reuse the data for certain activities. Consequently, they find this dump file, without knowing anything about it. The company has to buy a commercial data profiling software to re-understand the data, without the guarantee that the original schema can still be exported.

To help the enterprise (more generally, users) make sense of the data, data profiling automatically builds descriptive measures such as mean, variance, aggregates and additional metadata information including data type, uniqueness etc.

Data profiling activities are frequent and important for IT professional or data researchers. Early profiling can help clarify if the data source is correct. Without an overview of data, it is hard to advance on a project.

The benefits of data profiling are to improve the data quality, shorten the overall implementation cycle of a project and, more generally, improve users' understanding of data.

4.4.1 RDF data profiling

Recent work [1] has devised a web-based tool to profile arbitrary Linked Open Data (LOD) datasets. LOD datasets are usually represented in the form of RDF. Prominent examples including DBpedia, YAGO and Freebase have been extracting data from unstructured data. However, these knowledge bases are usually difficult to explore and understand. RDF profiling can help understand the structure of original dataset. A knowledge base may also evolve over time when more facts and entities are added to the knowledge base. This warrants the interest of re-examining the underlying dataset at a later point in time, to find out if and how it has changed.

There exist many commercial tools, such as IMB's Information Analyzer, for profiling relational datasets. However, there are few tools designed for Linked Open Data. Moreover, RDF is heterogenous, making it have a very different nature and call for specific profiling techniques.

The abovementioned paper proposed ProLOD++, a web-based tool for profiling LOD data, specifically RDF data. This tool is an extension of their previous work [5], which can profile RDF datasets by generating basic statistics including value patterns and distributions, predicate frequencies and some dependencies. The version ProLOD++ adds visualization functionality. It can, for example, identify predicate combinations that contain only unique value combinations to distinctly identify entities etc.

Our work, Dagger, integrated a data profiling module which can automatically detect the object data type of a predicate. The idea is: we explore all the different values for each predicate, parse each value using regular expressions, and infer the possible data type such as Strings, Floats and Integers. This is essential because, when applying aggregation functions to a measure (recall the data warehouse setting), the data type of measure should be specified. For instance, for numeric values, we can apply $max()$, $min()$, $sum()$ and $average()$, whereas for Strings type, we can only use $count()$.

4.4.2 A GUI tool for efficient RDF data analytics

A former intern in the Cedar team, namely Zheng ZHANG, had developed a GUI tool for exploring an RDF dataset. It can be used to explore different classes of an RDF graph, the number of different resources having a certain property, the number of different resources having a certain combination of properties etc. This tool can serve as an efficient tool for specifying a set of candidate facts (as we will explain in Chapter 6; more specifically, it can be used for property set pruning). It can also be seen as a tool for RDF profiling purposes. The tool can be accessed online at:

`https://perso.limsi.fr/zzheng/RDFDataVisualizationTool/bar_chart/`

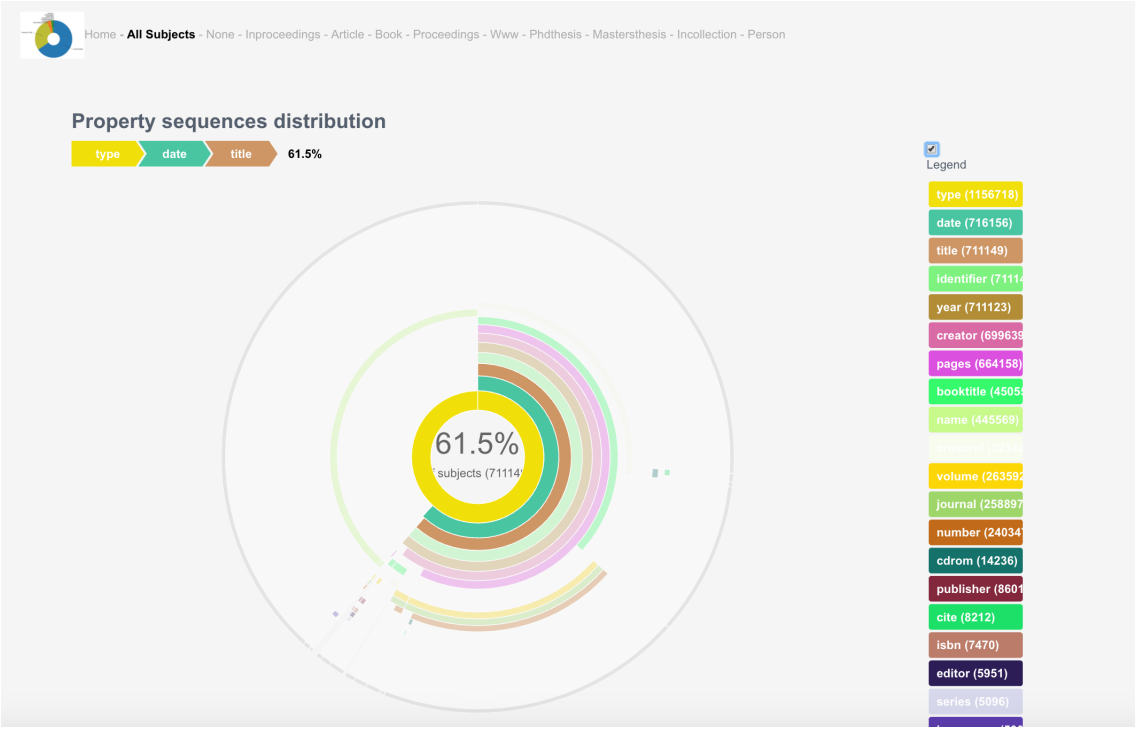


FIGURE 4.4: GUI tool for RDF property set analysis (all resources).

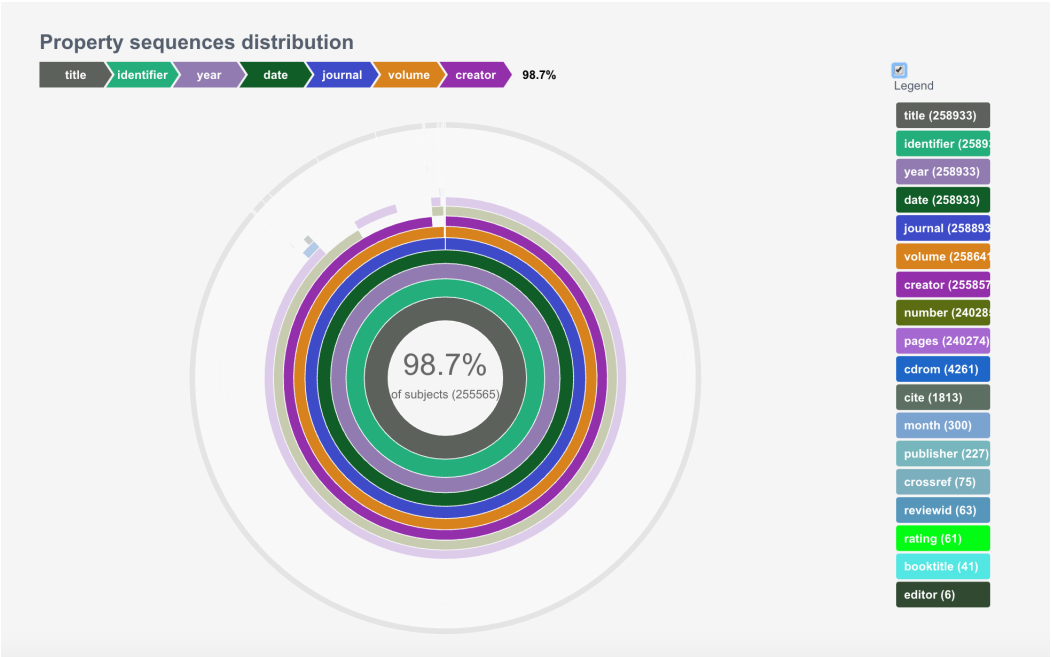


FIGURE 4.5: GUI tool for RDF property set analysis (resources of type *Article*).

Figure 4.4 and 4.5 show some screen captures of this profiling tool.

In the center of the tool interface, with the help of a JavaScript library, we can explore interactively each property of resources of a given type. A full colored circle corresponds to full support for a given property inside the set of facts of a given type, that is: 100% of resources of this type have a specific property (or set of properties). At right in the images, we see a grid bar for each property; the number specifies the

number of distinct resources (subjects) having this property. We can also navigate to other types by clicking on the corresponding type label on the top.

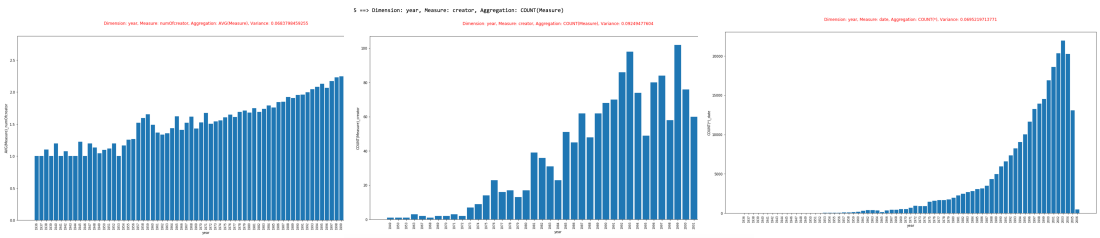


FIGURE 5.1: Interesting aggregates found in DBLP RDF data (1936-2006). From left to right: the average number of article authors, per year; the total number of book authors, per book publication year; the number of conferences in DBLP, per year.

5 Problem statement

RDF graphs may be very large and their structure complex and heterogeneous. In a bibliography database, authors may lack an e-mail, have one, or have several; some may also have a *fundedBy* property etc. This leads to a lack of pre-defined schema, and makes it complicated for users to find out the interesting information hidden in an RDF graph. We propose acquainting the user with an RDF graph G by presenting her *interesting aggregate queries* evaluated over G, and whose results are automatically laid out in the form of a two-dimensional diagram or bar chart (as exemplified in Figure 5.1). For instance, sample aggregate queries based on the RDF graph representing the DBLP bibliographic dataset are:

α_0 "For each year, the total number of distinct authors having published a conference paper that year"

α_1 "For each year, the average number of authors of the conference papers published that year"

We consider α_0 or α_1 *interesting* if there is some strong trend or interesting spike in the average number of authors along the time axis. The problem we consider can be stated as: *given an RDF graph G and an integer k, find the k most interesting aggregate queries over G.* We discuss these notions in Section 5.1.

5.1 Aggregate queries

Formally, an **RDF aggregate query** $q = \langle f, d, m, \oplus \rangle$ consisting of the following components:

1. A *set of facts*, or resources over which the aggregate is computed (in α_0 and α_1, the conference papers);
2. A *dimension* (for the time being only one; we plan to generalize to several dimensions in the future); (in α_0 and α_1, the year). Note that due to the heterogeneity of RDF, some facts may lack the dimension (for instance, a paper

may lack its publication year); in this case, that fact does not contribute to the aggregate query. Also due to RDF heterogeneity, some facts may have several values along the dimension, e.g., if the dimension is author affiliation and authors have multiple affiliations;

3. A *measure* (the authors in α_0 and the *number of authors* in α_1; the latter is not present in the RDF graph, but we *derive* such properties);
4. An aggregation function (*count(distinct(·))* in α_0 and *average* in α_1). The aggregation functions we consider are $\Omega = \{count, avg, sum, min, max\}$, possibly combined with a *distinct*. Again due to RDF heterogeneity, a fact may lack a measure, e.g., the institution may be unspecified for an author. In such cases: if the aggregation function is *count*, the fact contributes with a count of 0; otherwise, that fact does not contribute to the aggregate query.

A fact may also have several values for the measure, e.g. papers frequently have several authors; the aggregation function then applies over the complete set of measure values obtained for the facts having the same value for the dimension. Our queries are a subset of the RDF analytical queries introduced in [6, 2] and also of those expressible in W3C's SPARQL 1.1.

5.1.1 The semantics of aggregate queries

The semantics of an aggregate query specified as above is:

$$\{(x, \{\oplus(\{f.m\}|f.d = x)\}_x)\}$$

where:

- f ranges over all the facts
- $f.m$ is a value of the measure for the fact f
- $f.d$ is a value of the dimension for the fact f
- x is one possible value of $f.d$, in other words a grouping key
- $\{\oplus(\{f.m\}|f.d = x)\}_x$ is the result of the aggregation function evaluated over all the measure values for the facts having $f.d = x$

The query semantics, then, is a set of pairs of the form (grouping key, result of the aggregate for that grouping key).

5.2 Measure of Interestingness

Interestingness is currently defined as a certain measure over the set of values:

$$\{\oplus(\{f.m\}|f.d = x)\}_x$$

where $\{f.ms\}|f.d = x$ is the set of m values for all facts having the value x along dimension d.

Most of our experiments have relied on the *variance* (second statistic moment) measure. However, we have also implemented and experimented with *skewness* (third statistic moment) and kurtosis (the fourth moment); other measures of interestingness can be easily devised.

5.3 The Dagger Problem Statement

Based on the above notions, the problem addressed by Dagger can be stated as:

> Given an RDF graph, an interestingness measure and an integer k,
> find the k most interesting aggregate queries.

6 Architecture and algorithm

Here, we present Dagger's architecture (Section 6.1) and the concrete methods it employs to detect interesting aggregates: to select candidate fact sets (Section 6.2), candidate dimensions (Section 6.3), candidate measures (Section 6.4), candidate aggregation functions (Section 6.5).

6.1 Dagger's architecture

We have built Dagger as a Java and Python application on top of the PostgreSQL 9.6 relational database server; the GUI is based on Jupyter notebooks (http://jupyter.org). Dagger stores G as a collection of tables in PostgreSQL, one table for each class and property. While PostgreSQL bears the brunt of aggregate query evaluation, Dagger carefully translates between the RDF world and PostgreSQL, to faithfully reflect the RDF-specific features of our aggregation queries we explained above.

Dagger selects interesting insights through the following steps: **1.** identify several *candidate fact sets*; **2.** for each candidate fact set, explore a set of *candidate dimensions*; **3.** for each (f, d) pair, explore *candidate measures* m; **4.** for each (f, d, m) combination, pick *applicable* $\oplus$ functions; **5.** evaluate the interestingness of the queries resulting from steps 1. to 4. and retain those with top k values. In the sequel, we outline each step; for simplicity, we use f, d, m to refer to the respective queries *or* to their results on G.

6.2 Candidate fact sets

We identify two fact set enumeration methods.

1. If G contains some *rdf:type* triples, then we identify the set of class URIs in G (that is, the set of all URIs c such that a triple of the form $(x, rdf{:}type, c)$ exists in G), and for each such c, the set f_C of all resources stated to be of type c in G is a candidate fact set.
2. Given a user-specified support threshold t_{supp} between 0 and 1, for any set of properties $P = \{p_1, p_2, \ldots, p_n\}$ such that at least t_{supp} of the triple subjects in G have (at least once) each property in P, the set f_P of all resources having all the properties in P is a candidate fact set.

6.3 Candidate dimensions

A dimension should be a property which all facts in f have (those lacking it cannot contribute to q anyway), or *count(p)* for some property p which some facts in f have.

Further, d should have *relatively few distinct values*, as a property having almost a distinct value for each fact is not a meaningful agregation dimension. Formally, given a candidate fact set f and a property p which all facts from f have, let $S^0_{f,p}$ the set of distinct values of the property p on the facts from f. Property p is a candidate dimension if and only if $|S^0_{f,p}|/|f| \leq t_0$, where t_0 is a user-specified threshold t_0 between 0 and 1. For instance, in α_0 and α_1, *year* shows up as a candidate dimension as it has few distinct values (less than 100, while there are millions of publications).

Note: weak entities Weak entity cannot be used directly as dimensions; instead, they have to be combined with other owner entity. In our DBLP bibliographic example, resources of type *Article* have property like *volume*, which, however, is a weak entity and can only be interpreted combined with (within the context of) *journal*.

To handle such weak entity situations, we introduce a notion of *extended candidate dimension*: when encountering a weak entity, instead of using it as a dimension, the combination of the weak entity with the owner entity will be used as a dimension.

RDF graphs lack metadata information such that would enable us to detect weak entities, and we did not find works that automatically detect them. Thus, we assume that the weak entities and their "owner" entities are provided by the user in input to our tool, which accordingly builds *extended candidate dimensions*, pairing up weak entities with their owner entities.

Note: Almost-one attribute As we discussed before, due to the RDF heterogeneity, some facts in f may have several values along certain property p. Formally, given a candidate fact set f and a property p, let $S^1_{f,p}$ the set of distinct resources having more than one value for the property p in candidate fact set f. We call the property an *almost-one attribute (property)* if and only if $|S_{f,p}|^1/|f| < t_1$, where t_1 is a user-specified threshold between 0 and 1. For instance, *year* shows up as a almost-one attribute as every resource (fact) can have at most one value for it. Otherwise, it's not an almost-one attribute.

Note: Derived attribute If an attribute p is not an almost-one attribute, we *derive* an attribute p_{der} which counts the number of values of the property (i.e. $count(p)$). For instance, in our DBLP bibliography dataset, each journal article may have more than one value for the property *creator*, in which case, we derive an attribute which counts the number of creators for each journal paper.

6.4 Candidate measures

Given (f, d), a candidate measures m is a property sufficiently well supported among the facts in f (actually, those that do not have the property will not contribute to the aggregation result!)

Further, candidate measure should be different from candidate dimension, i.e, we are not interested in aggregate queries like "for each author, find the total number of distinct authors" as it is not meaningful.

The candidate measure should have no semantic derivation link connecting it with candidate dimensions. For instance, we are not interested in aggregate queries like "for each number of authors, find the total number of distinct authors": in this case, the dimension is a derived attribute from the measure, and the query is not meaningful.

For each m, we automatically detect the *majoritary data type* (string, integer, date, double) among all the values of property m for a fact in f. The details of the type detection method are given below.

6.4.1 A type detector for RDF data

In our work, the object data types that we considered is $T = \{Integer, Decimal, Date, String\}$. We propose an algorithm based on majority vote. Given a threshold t_{type}, if more than t_{type} of the object data type for a given property in a given candidate fact set is of type $t_p \subseteq T$, then we may infer that the object data type is t_p;

6.5 Candidate aggregation functions

Given (f, d, m), candidate aggregate functions $\oplus$ are chosen based on the type of the measure values. If this type is numeric, all Ω functions apply; *min*, *max*, *count* and *count(distinct)* apply to dates; *count* and *count(distinct)* apply to strings.

6.6 Ordering by interestingness

After formulating and evaluation aggregate queries, Dagger ranks the query results based on user-specified measure of interestingness (currently supported measures are: variance or second statistic moment, skewness or third statistic moment, and kurtosis or fourth statistic moment), and plot the top k most interesting query results.

Note: data normalization
As we discussed in Section 5.1.1, the semantics of a query is a set of pairs, specifying: one column corresponding different values of grouping key (in other words, the dimension) and another column corresponding to the aggregation result.

Each evaluation result of an RDF aggregate query may contains numeric aggregation results having a different domain range. For instance, aggregation results in α_0 (the number of distinct authors) may range to thousands, whereas in α_1, the aggregation results (the average number of authors) may range only between 0 and 10. The measure of interestingness thus calculated (for instance, variance) may range quite differently. To make them comparable, we need to *normalize* the original data distribution.

Different normalization mechanisms can be envisioned. In machine learning or in the business world, normalization usually means *feature scaling*, as follows. Given a value distribution, normalization scales each value into the range of $[0, 1]$ by subtracting the *min* and dividing by (*max* - *min*) of the distribution. This is a process of putting all the values into a common scale. Another notion close to normalization is that of *standardization*. Standardization rescales the values of original data so that they have a mean of 0 and a standard deviation of 1. This is quite useful when population parameters are known, and works well for datasets that follow a normal distributed.

We choose the feature scaling method, as follows. For each aggregate query results (in the form of a set of pairs), we scale each value of the aggregation result column into the range $[0, 1]$ and then we calculate e.g. the variance of normalized values. The rankings of aggregate queries are thus based on the variance of the normalized aggregation results, rather than on the direct results.

6.7 Algorithms

Algorithm 1 outline's Dagger's approach, in pseudocode.

Below, we first specify the input and output of the algorithm:

Input:

1. An RDF graph G, pre-loaded in a relational database management system;
2. Candidate fact set enumeration strategy $s \in \{s_0$: resources having a certain type, s_1: resources having a set of pre-specified supported properties$\}$;
3. If candidate fact set enumeration method is s_1, a set of pre-specified properties $P = \{p_1, p_2, \ldots, p_n\}$;
4. A user-specified support threshold t_{supp} between 0 and 1 (recall Section 6.2);
5. A user-specified threshold t_0 between 0 and 1 for dimension having few distinct values (recall Section 6.3);
6. A user-specified threshold t_1 between 0 and 1 for identifying almost-one attributes (and for producing derived attributes) (recall Section 6.3);
7. A user-specified threshold t_{type} between 0 and 1 to be used by the type detector (recall Section 6.4.1);
8. A set of weak entities W with their corresponding owner entities (recall Section 6.3);
9. User-specified preference of measure of interestingness $p_{inte} \in$ {Variance, Skewness, Kurtosis};
10. A user-specified integer k, corresponding to the number of desired ranked results.

Output: Top k most interesting aggregate queries.

The algorithm first takes a parameter s as input to specify the strategy of choosing candidate fact set, as we discussed in Section 6.2.

The procedure FindAggregateQueries(f_C) then takes candidate fact set f_C as input to find candidate dimensions and measures, as well as applicable aggregation functions based on measure data type (as we discussed in Section 6.4.1 and 6.5).

Next, the function MakeAggregateQueries(f_C, D, M, A) takes the **candidate fact set**, **candidate dimension set**, **candidate measure mapping** (mapping each dimension to a set of applicable candidate measures) and **candidate aggregation function mapping** (mapping each candidate measure, along with its candidate dimension, to a set of applicable candidate aggregation functions) as input to issue a set of aggregate queries that we formally introduced in Section 5.1. The formulation structure is illustrated in Figure 6.1, specifying: we take one candidate dimension d from the candidate dimension set D, then we take one measure m from the mapping M, and at last, we take one aggregation function $\oplus$ from the mapping A. One aggregate query $q = \langle f, d, m, \oplus \rangle$ is thus formulated.

The function **ExtendWeakEntity(p, c)** is executed whenever we encounter a weak entity. For instance, for *journal article*, we have the property *volume*, which is a weak entity, and should be combined with *journal*. As we discussed in Section 6.3, no existing work can automatically detect the weak entities in RDF graphs, we thus

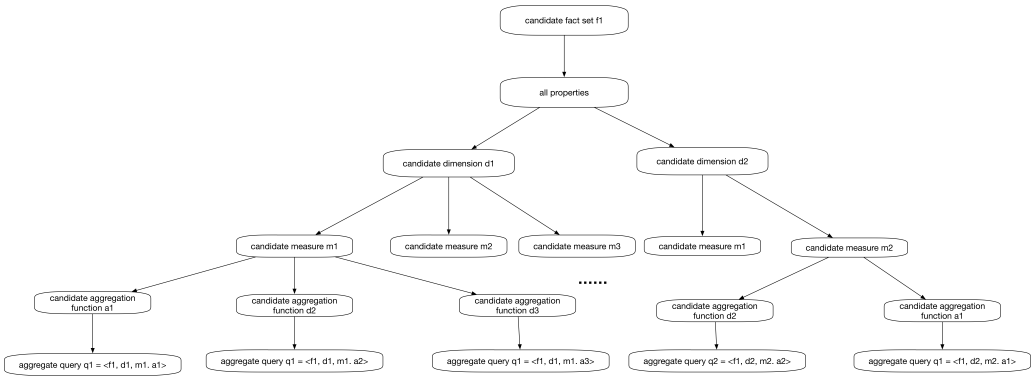


FIGURE 6.1: Making aggregate queries based on f, D, M and A.

take the weak entity information as an input of the program. Specifically, the user must provide an input file specifying all the weak entities in an RDF graph, along with their owner entities. Dagger builds *extended candidate dimensions* based on this information. The weak entity input file contains mainly three field as illustrated in Figure 6.2, separated by tabs. The first field corresponds to the resource type, which can be, for instance, *article*, *conference paper* or *book* etc. (this type should be specified by the full URI appearing in the original RDF dataset). The second field is the weak entity, for instance *volume* for resource type *article*. The last field is an *ordered* list of owner entities, *the top of the hierarchy to the lower hierarchy, separated by a comma, without any space.* For some weak entities such as *volume*, there is only one owner entity *journal*, whereas for others like *issue number*, its owner entity should be the combination of *journal* and *volume* (in which case, the owner entity *journal* appears at the top of the hierarchy, followed by *volume*, and at last the *issue number*). For instance, given a issue number '101', we need to know not only the volume number (38, for instance), but also the journal name ('SIGMOD Record', for instance). In this case, the extended candidate dimension is {journal + volume + issue number} ('SIGMOD Record-38-101', for instance). Figure 6.3 gives an example of the input file of weak entities, specified on the DBLP dataset previously discussed.

field 1	field 2	field 3
resource type	weak entity	ordered owner entities

FIGURE 6.2: Specification of weak entity input file

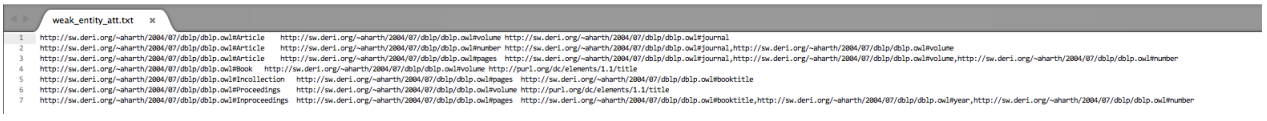


FIGURE 6.3: An example of the input file of weak entities, used in DBLP dataset

Finally, the algorithm evaluates the generated aggregate queries and ranks them based on the user-specified measure of interestingness p_{inte}, as described in Section 6.6. The algorithm outputs the top k most interesting aggregate queries.

Algorithm 1 Dagger's core algorithm

$D \leftarrow$ an empty set to store candidate dimensions
$M \leftarrow$ an empty map to store candidate measures, mapping each candidate dimension to a set of candidate measures
$E \leftarrow$ an empty set of derived attributes
$A \leftarrow$ an empty map to store candidate aggregation functions, mapping each measure to a set of applicable aggregation functions, along with the candidate dimension
$Q \leftarrow$ an empty set to store candidate aggregate queries
if $s = s_0$ **then**
    for all type(class) c in RDF graph G **do**
        Candidate fact set $f_C \leftarrow$ the set of all resources stated to be of type c
if $s = s_1$ **then**
    boolean supported $\leftarrow$ True
    for each property p in P **do**
        $S_{p,G} \leftarrow$ the set of distinct resources having the property p in G
        if $|S_{p,G}|/|G| \leq t_{supp}$ **then**
            supported $\leftarrow$ False
            break
    if supported = True **then**
        Candidate fact set $f_C \leftarrow$ the set of all resources having each property in P
$Q \leftarrow$ MakeAggregateQueries(f_C, D, M, A)
$Q_{ranked} \leftarrow$ EvaluateAggregateQueries(Q, p_{inte})
return OutputTopKAggregateQueries(Q_{ranked}, k)
procedure FindAggregateQueries(f_C)
    for each property p in f_C **do**
        $S^0_{f,p} \leftarrow$ set of distinct values of the property p of the facts from f_C
        $S^1_{f,p} \leftarrow$ set of distinct resources having the property p in f_C
        if $|S^0_{f,p}|/|f_C| \leq t_0$ **and** $|S^1_{f,p}|/|f_C| \geq t_{supp}$ **then**
            IsSupported(p) $\leftarrow$ True
            if $p \in W$ **then**
                $p \leftarrow$ ExtendWeakEntity(p, c)
            AddCandidateDimension(p, D)
            $S^2_{f,p} \leftarrow$ set of distinct resources having more than one value for p
            if $|S^2_{f,p}|/|f| \leq t_1$ **then**
                IsAlmostOneAttribute(p) $\leftarrow$ True
            else
                $p_{der} \leftarrow$ MakeDerivedAttribute(p, f)
                IsDerivedAttribute(p_{der}, p) $\leftarrow$ True
                AddCandidateDimension(p_{der}, D)
                AddDerivedAttribute(p_{der}, E)
    for each candidate dimension d in D **do**
        for each property p in f_C **do**
            if IsSupported(p) = True **and** $p \neq d$ **then**
                if IsDerivedAttribute(d, p) = False **then**
                    AddCandidateMeasure(p, d, M)
        for each derived attribute p_{der} in E **do**
            if $p_{der} \neq d$ **then**
                AddCandidateMeasure(p_{der}, d, M)
        for each measure m in M **do**
            $m_{type} \leftarrow$ InferMeasureType(m, t_{type})
            $A \leftarrow$ InferApplicableAggregatesBasedOnType(m, m_{type})

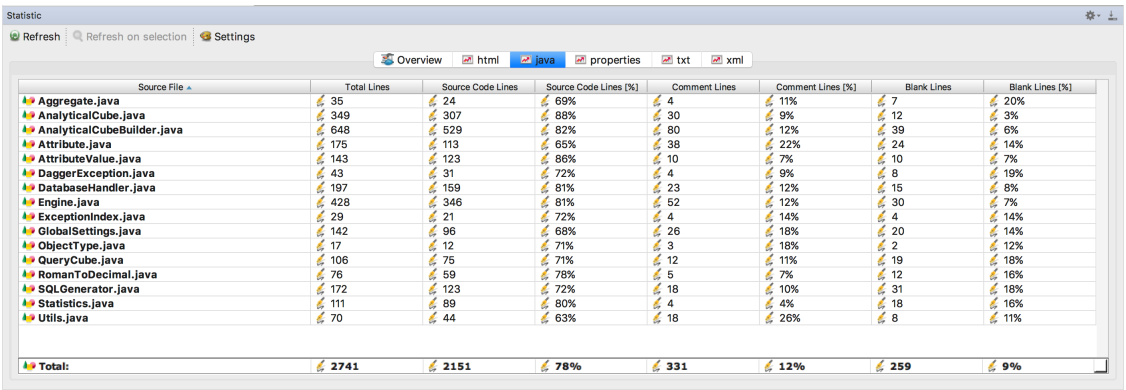
7 Implementation and results

7.1 Dataset used for experiments

Our experiments have mostly been carried over a subset of the DBLP bibliographic dataset (`http://dblp.uni-trier.de/`); it has about 8 million triples. Other RDF data sets can be used, in particular for the ISWC demo, we plan to use other real datasets (DBLP, Linked Clinical Trials, BBC Music Pro- grams, DBPedia etc.) as well as synthetic ones (LUBM [10] and BSBM [3] benchmark data).

7.2 Software used in the project

We build Dagger as an application using Java (version 1.8) and Python (version 3.6); the project has approximately 2700 lines of Java code and 100 lines of python code. Figure 7.1 illustrates some basic statistics about Java code of Dagger(generated by IDE IntelliJ IDEA):



Source File	Total Lines	Source Code Lines	Source Code Lines [%]	Comment Lines	Comment Lines [%]	Blank Lines	Blank Lines [%]
Aggregate.java	35	24	69%	4	11%	7	20%
AnalyticalCube.java	349	307	88%	30	9%	12	3%
AnalyticalCubeBuilder.java	648	529	82%	80	12%	39	6%
Attribute.java	175	113	65%	38	22%	24	14%
AttributeValue.java	143	123	86%	10	7%	10	7%
DaggerException.java	43	31	72%	4	9%	8	19%
DatabaseHandler.java	197	159	81%	23	12%	15	8%
Engine.java	428	346	81%	52	12%	30	7%
ExceptionIndex.java	29	21	72%	4	14%	4	14%
GlobalSettings.java	142	96	68%	26	18%	20	14%
ObjectType.java	17	12	71%	3	18%	2	12%
QueryCube.java	106	75	71%	12	11%	19	18%
RomanToDecimal.java	76	59	78%	5	7%	12	16%
SQLGenerator.java	172	123	72%	18	10%	31	18%
Statistics.java	111	89	80%	4	4%	18	16%
Utils.java	70	44	63%	18	26%	8	11%
Total:	**2741**	**2151**	**78%**	**331**	**12%**	**259**	**9%**

FIGURE 7.1: Some statistics about Java code of Dagger

Dagger runs on top of an RDBMS; in particular, we used a PostgreSQL 9.6 relational database server.

The aggregate queries are formulated by a Java program. The queries thus generated are then evaluated by PostgreSQL, a Java program fetches the query evaluation results and computes their interestingness.

The top k most interesting aggregate query results are then exported by Java to the file system, in the TSV (tab-separated values) format.

The visualization module is implemented in Python. The packages we have used include: **Pandas** (a famous open source providing high-performance, easy-to-use data structures and data analysis tools for the Python programming language); **Numpy**

(a fundamental package for scientific computing with Python) and **matplotlib** (a Python 2D plotting library which produces publication quality figures in a variety of hardcopy formats and interactive environments across platforms).

We used **Jupyter notebook** for plotting interesting bar charts. Jupyter Notebook is a web application that allows to create and share documents that contain live code, equations, visualizations and explanatory text in an interactive way. It is frequently used in the Data Science communities.

7.3 Running environment

Most of the tests of implementation have been performed on my local computer (MacBook Pro, MacOS 10.12.6, CPU 2.5GHz Intel Core I7, RAM 16G).

We also tested our program on a machine in the **Cedar cluster**. One cluster has totally 40 processors, each of them has a Intel(R) Xeon(R) CPU E5-2640 v4 @ 2.40GHz processor. The total RAM of a cluster is nearly 132GB. The default Linux distribution is 3.10.0.

7.4 Results: interesting aggregate queries found

Figures 7.2 to 7.13 show some interesting aggregate queries found by Dagger in our DBLP dataset.

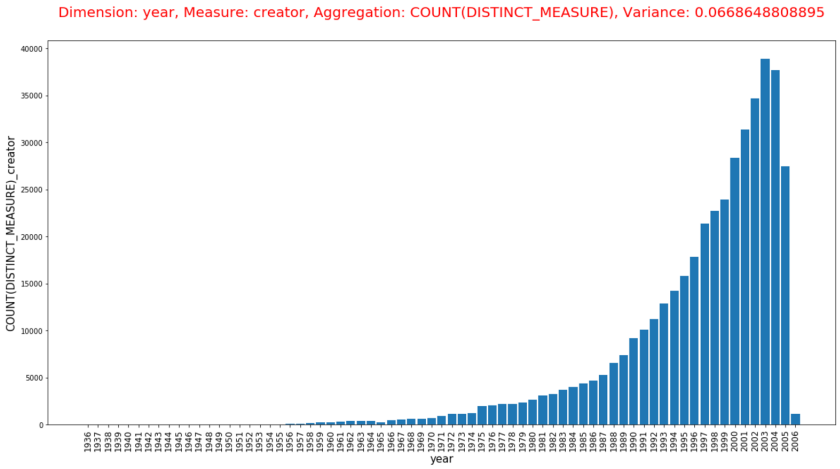


FIGURE 7.2: Sample plot, resource type is *journal article*, dimension is *year*, measure is *creator* and the aggregation function is *count(distinct)*. This query counts the number of creators of journal articles per year in DBLP dataset.

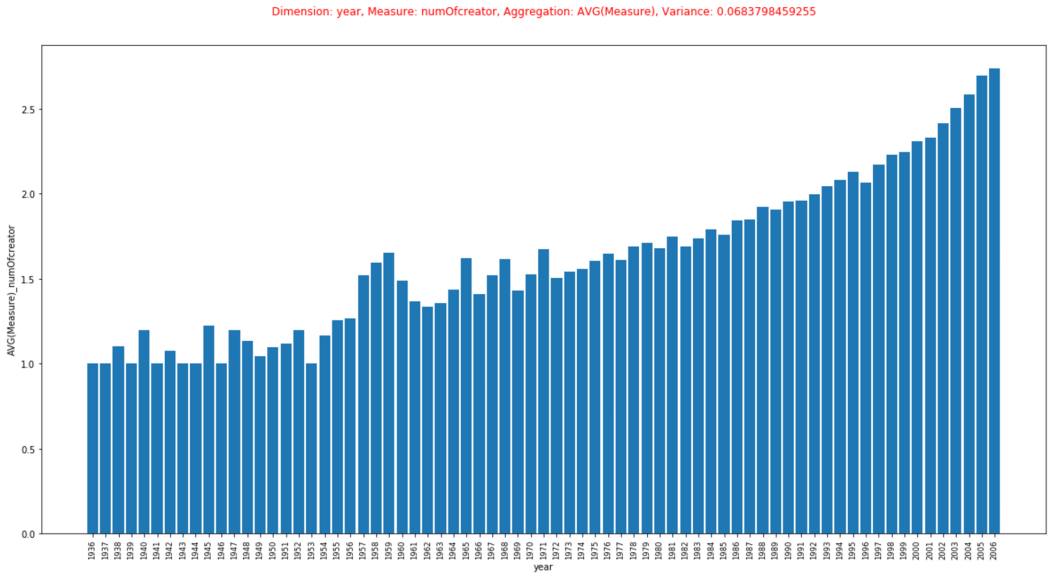


FIGURE 7.3: Sample plot, resource type is *journal article*, dimension is *year*, measure is a derived attribute *number of creators* and the aggregation function is *avg()*. This query counts the average number of creators of journal articles per year in DBLP dataset.

7.5 Demonstration video and code repository

We have submitted this work to the International Semantic Web Conference (ISWC-2017) as a demo paper and it has been accepted. A video of our demo appears at https://team.inria.fr/cedar/projects/dagger.

Source code can be found at https://gitlab.inria.fr/sshang/dagger.

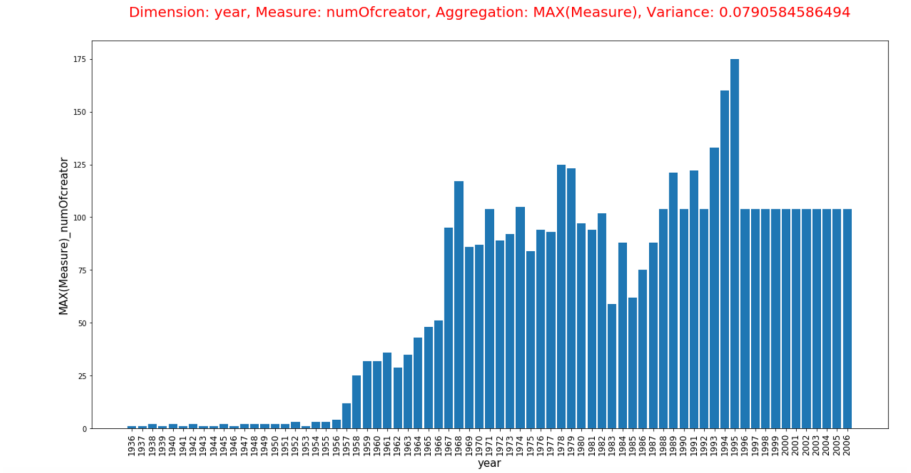


FIGURE 7.4: Sample plot, resource type is *journal article*, dimension is *year*, measure is *number of creators* and the aggregation function is *max()*. This query counts max number of creators of journal articles per year in DBLP dataset. We can see that 100 shows up as a ceiling after 1996.

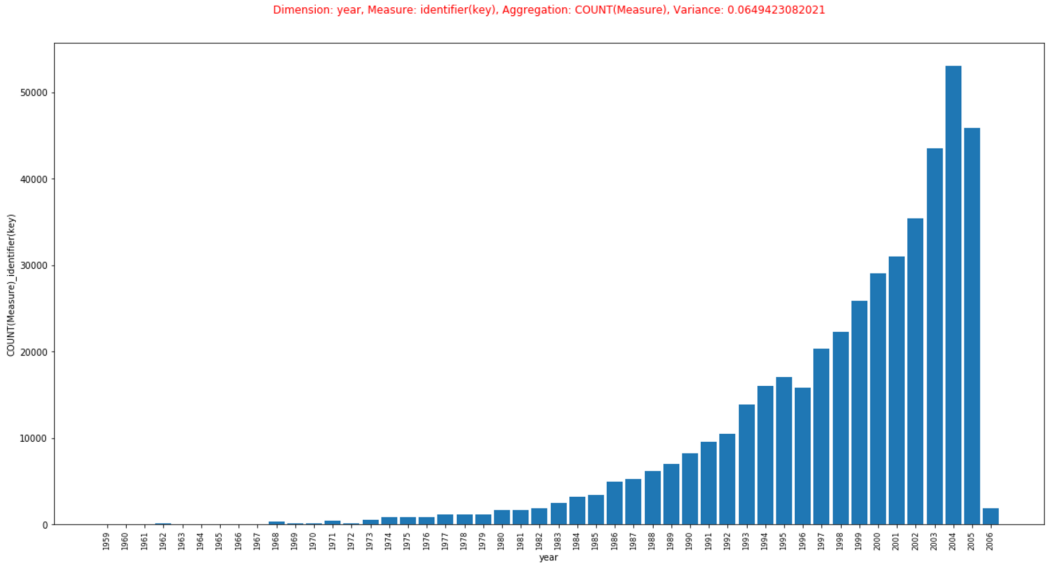


FIGURE 7.5: Sample plot, resource type is *conference paper*, dimension is *year*, measure is *identifier* and the aggregation function is *count()*. This query counts number of conference papers per year in DBLP dataset.

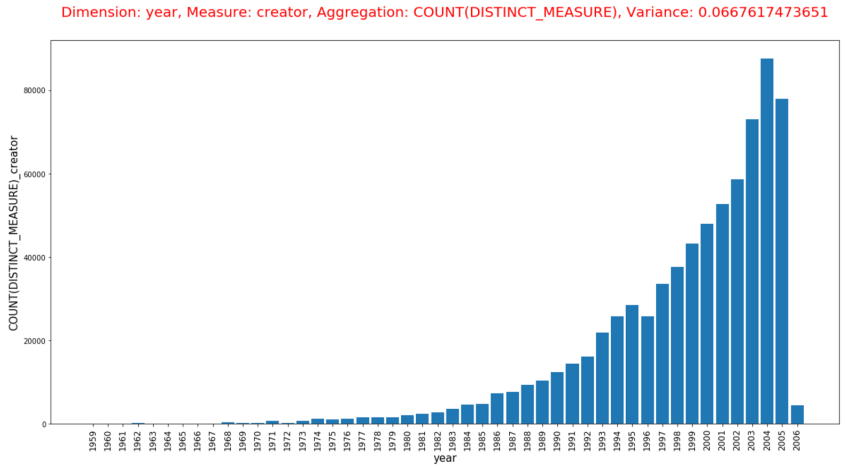


FIGURE 7.6: Sample plot, resource type is *conference paper*, dimension is *year*, measure is *creator* and the aggregation function is *count(distinct)*. This query counts the number of authors (creators) of conference papers per year in DBLP dataset.

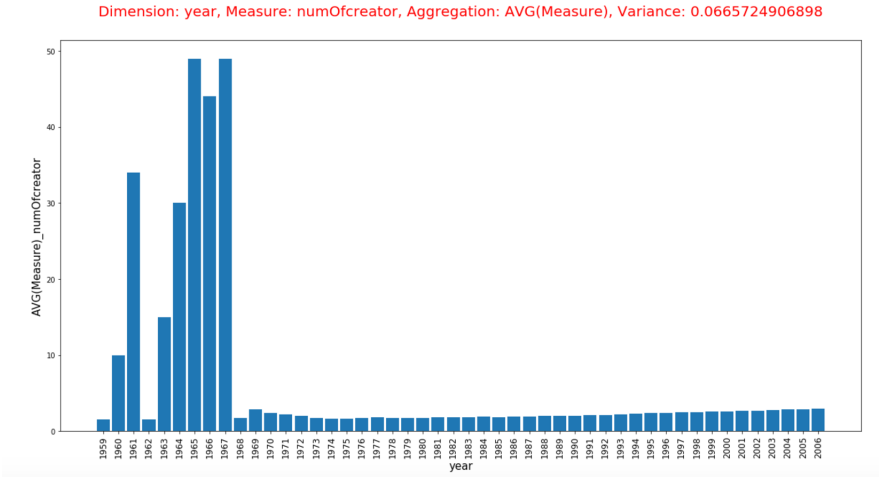


FIGURE 7.7: Sample plot, resource type is *conference paper*, dimension is *year*, measure is *number of creators* and the aggregation function is *avg()*. This query counts the average number of authors (creators) of conference papers per year in DBLP dataset. We see that there is a sudden drop in 1968, and the number of authors remains less than 10 afterwards.

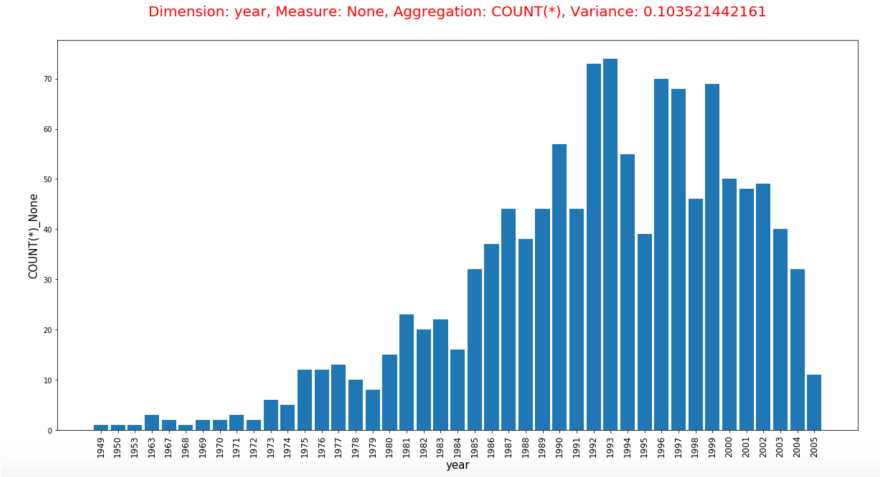


FIGURE 7.8: Sample plot, resource type is *book*, dimension is *year*, measure is *None* (without a measure) and the aggregation function is *count()*. This query counts number of books published per year in DBLP dataset.

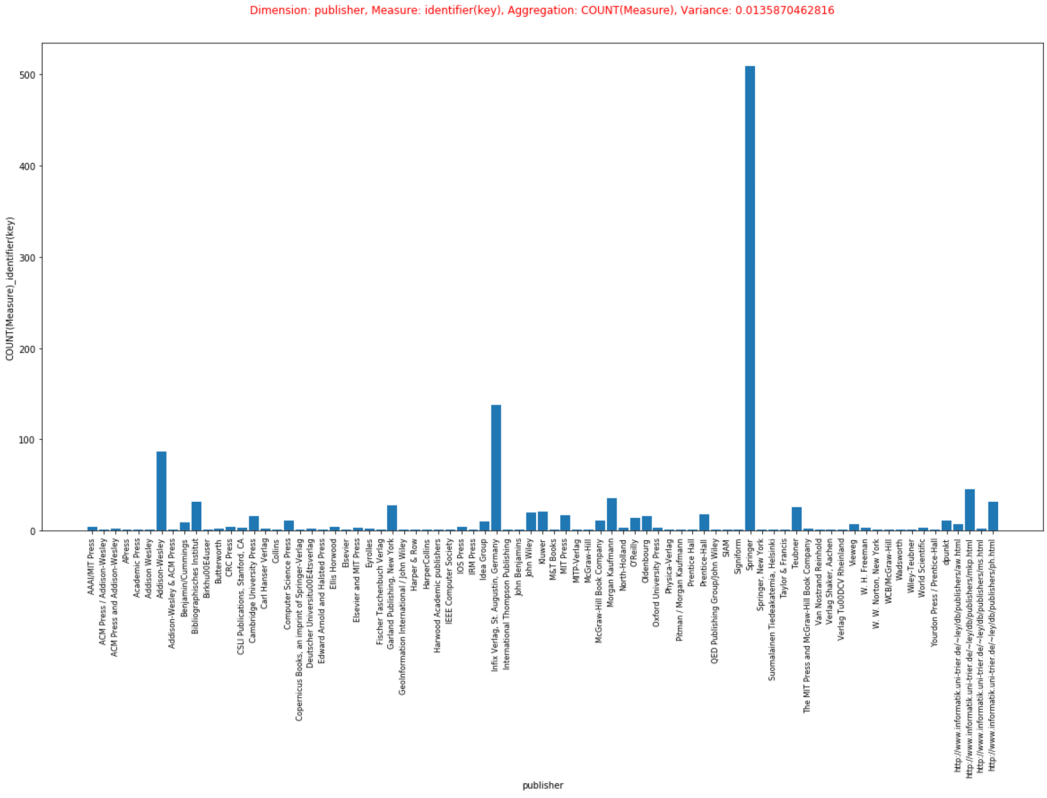


FIGURE 7.9: Sample plot, resource type is *book*, dimension is *publisher*, measure is *identifier* and the aggregation function is *count()*. This query counts number of books published by each publisher in DBLP dataset. We can see that publisher *Springer* shows up like a spike.

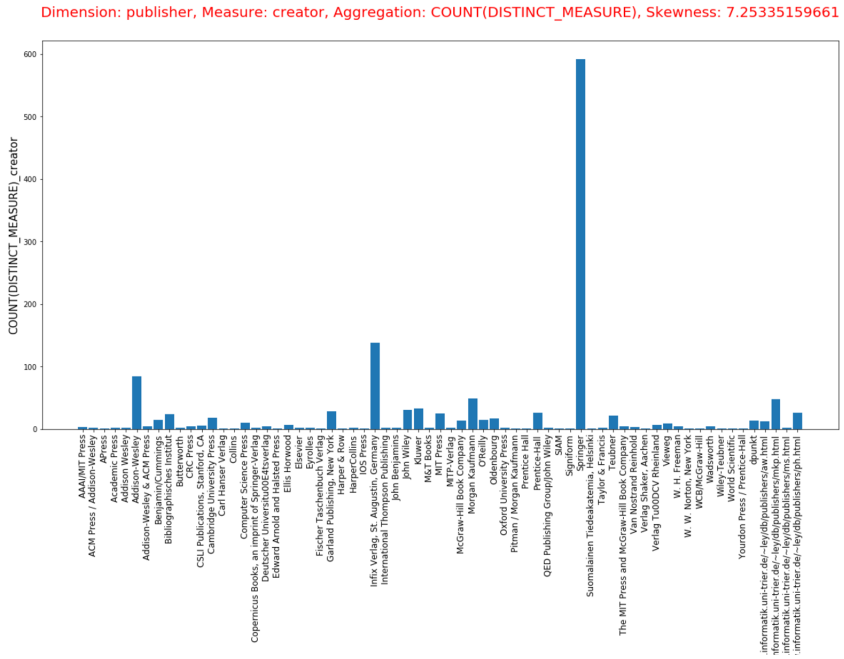


FIGURE 7.10: Sample plot, resource type is *book*, dimension is *publisher*, measure is *creator* and the aggregation function is *count(distinct)*. This query counts number of authors having published works on each scientific publisher in DBLP dataset. We can see that publisher *Springer* also shows up like a spike. This query shows up on the top if we use skewness as the measure of interestingness.

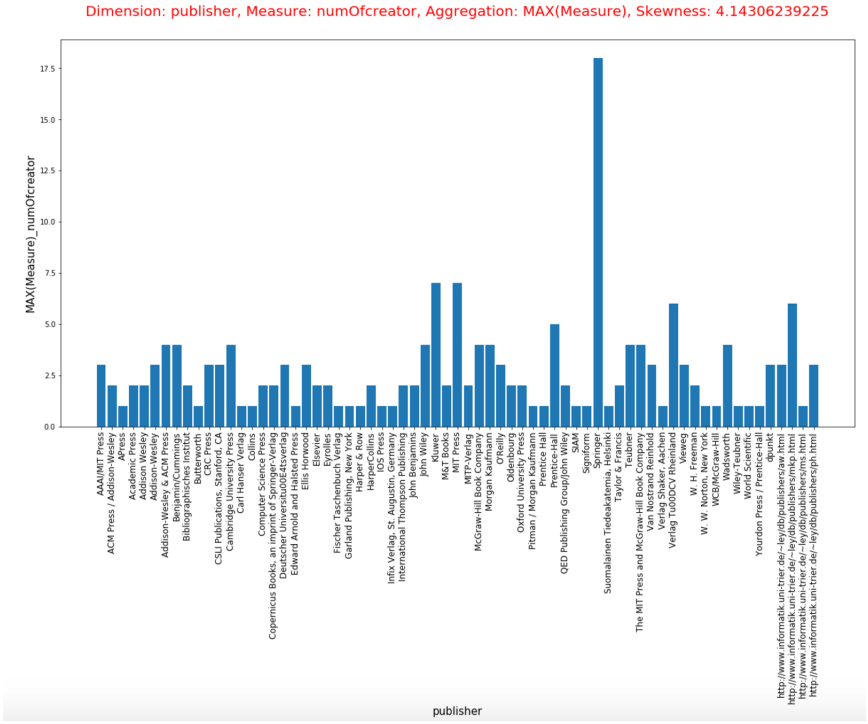


FIGURE 7.11: Sample plot, resource type is *book*, dimension is *publisher*, measure is *number of creators* and the aggregation function is *max()*. This query counts max number of authors having published works together on each scientific publisher in DBLP dataset. We can see that publisher *Springer*, again, shows up like a spike.

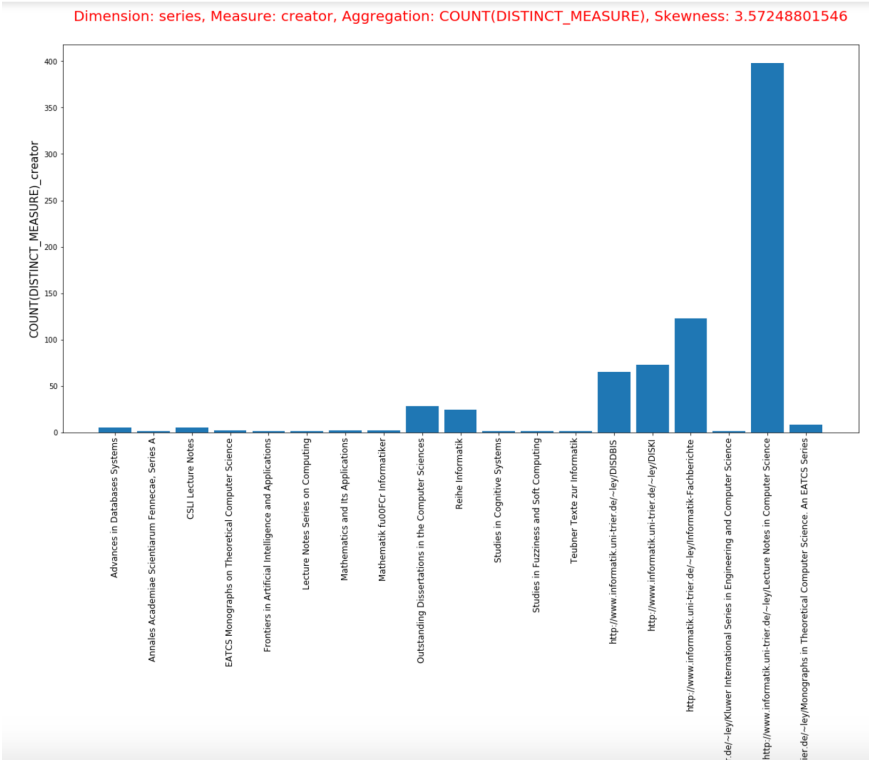


FIGURE 7.12: Sample plot, resource type is *book*, dimension is *series*, measure is *creator* and the aggregation function is *count(distinct)*. This query counts number of authors having published works on each series (the dimension) in DBLP dataset. We can see that the series *Lecture notes in Computer Science* shows up.

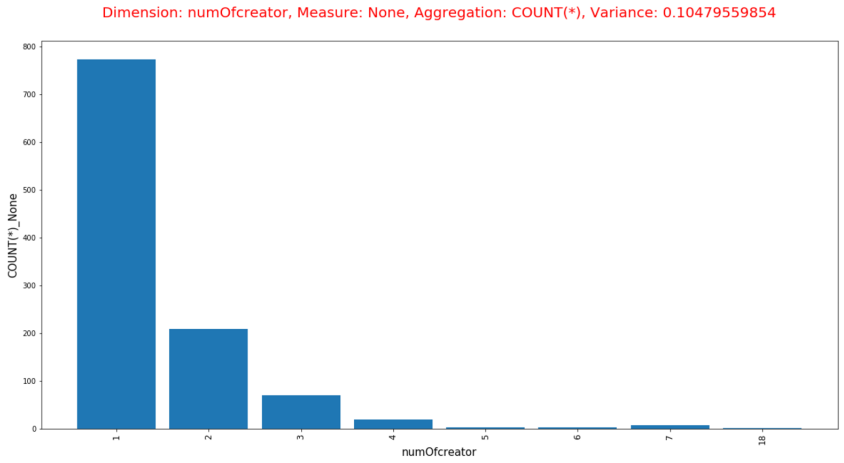


FIGURE 7.13: Sample plot, resource type is *book*, dimension is *number of creators*, measure is *None* (without a measure) and the aggregation function is *count()*. This query gives us an insight that most books published only have one author.

8 Conclusion

8.1 Perspectives for future work

As my internship will end in early September 2017, several ways to continue this work can be envisioned.

First of all, as we discussed in Section 4.3, the exploratory framework has not been integrated into Dagger yet. Further work includes devising an exploratory framework for providing more aggregate queries that fit a user's (or domain expert's) need. For instance, machine learning models may need to be integrated to build a concrete user model to reduce the data exploration space.

Secondly, this work only takes one dimension (actually, for weak entities, several are combined, but there result is still a single dimension of analysis) and one measure into account. Further work may include extending the framework to multiple dimensions and multiple measures. The aggregation function pruning method may still need to be improve. For now, we only take almost-one characteristic, weak entity, and measure data type into account.

Then, for the weak entity, instead of asking the user to specify them, one could envision building an algorithm which automatically detects such situations. This topic has already been well understood in the database community, and it should be useful to map it to RDF data; this would be an interesting avenue for continuing work on RDF data profiling.

Next, an interactive interface should be designed, as this tool serves as a data analytics tool for end-user, although domain expert may feel easy to use it, but it is still difficult to use for general users. A user case study should also be done. For data visualization part, instead of using static 2D bar chart, an interactive interface will be more interesting. Packages like matplotlib and seaborn have already these kinds of implementations and they can be easily integrated to a web-based application.

At last, performance is an important topic but it has not been much considered during this early work. The current algorithm evaluates all the eligible agregation queries, in a rather exhaustive way. Further work may include query optimization, computation and result sharing etc. Integrating this tool in a distributed data processing framework like Spark could also be useful, as Spark has provided a full tool set for SQL-like queries and data analytics.

8.2 Some reflections

In this part, I note some personal reflections based on this internship.

8.2.1 A general retrospect

During this internship of more than 5 months, we devised and implemented **Dagger**, an efficient tool for RDF analytics based on aggregate queries.

In the beginning of the internship, I mainly focused on the reading of previous work, as it is introduced in Chapter 4. Along with the reading, I wrote several summaries of the papers that I have read. This period was very important for me, it led me to go into the details of the problem and helped me have a integral understanding of the related work.

Each week, I had a meeting with my advisor Yanlei Diao and Ioana Manolescu. After the meeting, I used to write meeting notes. My advisor Ioana has emphasized several times the importance of meeting notes. For my part, the notes really helped me a lot to recall the details of the meeting. Moreover, it also helped me a lot to write this final report. During the meeting, we discussed the previous work, the related ideas, and tried to propose a new idea based on the previous work. At around the middle of my internship, a basic skeleton of the algorithm was designed. I then wrote the pseudocode based on the idea.

During the next period, I mainly focused on the implementation of the idea. I used Java as the main implementation language of the core algorithm and Python for data visualization. Lots of new ideas have also been proposed during the implementation part, including aggregate pruning, almost-one attribute, weak entity etc.

At around the end the June, a basic prototype of Dagger has been implemented. I have encountered some difficulties during the implementation part. Firstly, there exist some tools in `cedar` maven repository (`https://gitlab.inria.fr/cedar`), it will be better to reuse these existing tools for facilitating the implementation. For now, only the `cedar-commons package` was used in the project, for connecting the database. Further work can integrate the dictionary-encoding repository etc. Secondly, the structure of the code has been re-designed two times, for facilitating the integrating of new features (for instance, multiple dimensions and measures can be easily implemented by extending existing structure). More details have been well commented along the code. Next, exceptions are not well handled in the early stage. As Dagger has many operations over the database, the reason for some exceptions are not clear because they may be concerned with the configuration of the database server. A more complete mechanism for handling exceptions have been designed later.

In July and August, we focused mainly on some improvements. We also submitted our work to the JDSE2017 (the Junior Conference on Data Science and Engineering). It was accepted in the demo session. Moreover, this work has also been submitted to ISWC2017 (the International Semantic Web Conference, rank A) and was accepted as a demo paper in the end of August. The reviewers have provided some very useful suggestions. For instance, one reviewer signaled a quite interesting paper [7] about formal specifications of what is "interesting". This should be taken into the consideration in our further work.

We plan to attend ISWC 2017 in Vienna during 21the to 25th October to present this work.

During the internship, lots of knowledge I have learned from school has been used. For instance, the data warehouse and Semantic Web course are both important background for this internship. Knowledge about database systems also played an important part. The various engineering courses and project experience in Polytech Paris-Sud helped me advance the implementation quickly.

8.2.2 Research environment vs. industry

A research environment like Inria has been a totally new place for me. Prior to that, I have an experience working in a french software company called ICDSC in 2016. Here, I will make a summary based on my working experience, especially the differences between working in a company and a research institution.

First of all, from a administrative perspective, Inria and the former company have different hierarchical structures. As we introduced before, research works in Inria are carried out by "project teams". In a team, people with similar academic background gathered to pursue challenging research problems. The research environment has a rather flat hierarchy. In contrast, industrial companies usually have a more complex hierarchical structure such as project manager, team leader, general manager etc.

Secondly, from a team perspective, research team has more flexibility than company project team. For instance, project team of Inria can devise research objectives based on their own interests, new ideas usually come up during the discussion and they can be soon tested by experiments. In contrast, a project team in a company has to follow a basic procedure and then advance progressively. New ideas may be ignored if they cannot meet company's short-term objective. Although many methods can be proposed, for instance agile software development (in the IT domain), a project team in a company may still encounter many difficulties from different aspects, including administration, financing, human resources etc.

Next, from a financial perspective, research works have to be carried out based on the financial budget of the team. Inria has a total budget of €231M in 2016. As far as I am concerned, funding is extremely important for the advancing of research work. More importantly, the funding should not base on some profitable target, as only a few research works can have a direct influence on the industry. Moreover, research work usually takes the lead of the technology, meaning that it takes some time to industrialize it. However, commercial projects in company have to meet some profitable objective, either from customers or from indirect benefits like advertising.

Last but not least, from an engineering and research perspective, research works focus more on problem discovering, new solutions, theoretical proving, rather than engineering details. For example, a prototype is more important than a solid software for research work in computer science, as not all ideas are proved to be feasible in the beginning. However, in a company, industrial product has to be carefully designed in the very beginning, including large amounts of engineering problems, like the choice of technology, the feasibility and reliability etc. As I have followed both an engineering education and a research-related master program, I am well positioned to see the differences. For instance, the final goal of my research project has not been very clear in the very beginning. We had to follow lots of previous work, discuss and then try to propose a new problem, devise a new approach and experiment the feasibility. It took me about one month to get into the detail of the problem and then go further.

8.3 Conclusion

During this internship, I have been studying into different topics, including semantic web, RDF, OLAP operations, data cubes, data profiling, database, SQL and various relative topics like statistics, machine learning and data visualization. The implementation part also helped me improve my programming skills. This internship helped me to better understand these topics, they should be very useful in my future career.

My internship advisor and academic advisors both gave me a lot of help, including methodology for paper reading, methodology of research work and attitude towards professional work etc. I learned much from them. I am very grateful for being given this opportunity.

I have encountered lots of difficulties during this internship. In the beginning of my internship, as I am originally from an engineering school, the research environment was a totally new place for me. It took me significant effort to follow the research workplace procedures. With the help of my advisors, I have been advancing gradually and finally reached some goals. This made me feel confident and grateful.

This internship also helped me to better understand the scientific world. Inria-Saclay is a wonderful place to carry out research work. A lot of senior researchers have gathered here. They are all invaluable partners for junior researchers and students.

Bibliography

[1] Ziawasch Abedjan et al. “Profiling and mining RDF data with ProLOD++”. In: *Data Engineering (ICDE), 2014 IEEE 30th International Conference on*. IEEE. 2014, pp. 1198–1201.

[2] Elham Akbari Azirani et al. “Efficient OLAP operations for RDF analytics”. In: *ICDE Workshops*. 2015. DOI: 10.1109/ICDEW.2015.7129548. URL: https://doi.org/10.1109/ICDEW.2015.7129548.

[3] *Berlin SPARQL Benchmark (BSBM)*. http://wifo5-03.informatik.uni-mannheim.de/bizer/berlinsparqlbenchmark/.

[4] Dritan Bleco and Yannis Kotidis. “Entropy-based Selection of Graph Cuboids”. In: *GRADES*. 2017. DOI: 10.1145/3078447.3078449. URL: http://doi.acm.org/10.1145/3078447.3078449.

[5] Christoph Böhm et al. “Profiling linked open data with ProLOD”. In: *Data Engineering Workshops (ICDEW), 2010 IEEE 26th International Conference on*. IEEE. 2010, pp. 175–178.

[6] Dario Colazzo et al. “RDF analytics: lenses over semantic graphs”. In: *WWW*. 2014. DOI: 10.1145/2566486.2567982. URL: http://doi.acm.org/10.1145/2566486.2567982.

[7] SIMON COLTON, ALAN BUNDY, and TOBY WALSH. “On the notion of interestingness in automated mathematical discovery”. In: *International Journal of Human-Computer Studies* 53.3 (2000), pp. 351 –375. ISSN: 1071-5819. DOI: http://dx.doi.org/10.1006/ijhc.2000.0394. URL: http://www.sciencedirect.com/science/article/pii/S107158190090394X.

[8] Kyriaki Dimitriadou, Olga Papaemmanouil, and Yanlei Diao. “AIDE: an active learning-based approach for interactive data exploration”. In: *IEEE Transactions on Knowledge and Data Engineering* 28.11 (2016), pp. 2842–2856.

[9] David C Faye, Olivier Curé, and Guillaume Blin. “A survey of RDF storage approaches”. In: *Revue Africaine de la Recherche en Informatique et Mathématiques Appliquées* 15 (2012), pp. 11–35.

[10] Yuanbo Guo, Zhengxiang Pan, and Jeff Heflin. “LUBM: A benchmark for OWL knowledge base systems”. In: *J. Web Sem.* 3.2-3 (2005), pp. 158–182.

[11] Amit Krishna Joshi, Pascal Hitzler, and Guozhu Dong. “Logical linked data compression”. In: *ESWC*. 2013.

[12] Shahan Khatchadourian and Mariano P Consens. “Understanding billions of triples with usage summaries”. In: *Semantic Web Challenge* (2011).

[13] Felix Naumann. “Data profiling revisited”. In: *ACM SIGMOD Record* 42.4 (2014), pp. 40–49.

[14] Eric Prud’hommeaux and Andy Seaborne. *SPARQL Query Language for RDF*. W3C Recommendation. 2008. URL: http://www.w3.org/TR/rdf-sparql-query/.

[15] Bo Tang et al. “Extracting Top-K Insights from Multi-dimensional Data”. In: *SIGMOD*. 2017. DOI: 10.1145/3035918.3035922. URL: http://doi.acm.org/10.1145/3035918.3035922.

[16] Manasi Vartak et al. "SEEDB: Efficient Data-Driven Visualization Recommendations to Support Visual Analytics". In: *PVLDB* 8.13 (2015). URL: http://www.vldb.org/pvldb/vol8/p2182-vartak.pdf.